A Hybrid BVH Structure for Interactive GPU Ray Tracing

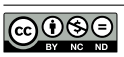
Andrew Bauer
Walt Disney Animation Studios
Burbank, USA
andrew.bauer@disneyanimation.com

Yining Karl Li
Walt Disney Animation Studios
Burbank, USA
karl.li@disneyanimation.com

Figure 1: Our hybrid BVH structure combines elements of a two-level BVH and a Mega-BLAS BVH to allow for efficient interactive ray tracing of complex production scenes, such as this one from "Zootopia 2", with active geometric editing and real-time animation playback. Left: Visualization of unique bottom-level acceleration structures. Middle: Beauty render. Right: Visualization of unique scene objects.

Abstract

Choosing a bounding volume hierarchy (BVH) structure for accelerating ray tracing applications typically involves trading off between BVH build/update speed and BVH traversal speed. Two-level BVHs offer faster build/update speeds but lower traversal performance and can still have degraded build speed in pathological cases common to production scenes, while monolithic single-level BVHs provide ideal traversal performance but suffer from slower worst-case build speeds. We introduce a *hybrid approach* that combines the strengths of both while minimizing their downsides. Our system places instanced objects into a two-level BVH, while non-instanced objects are partitioned into three separate monolithic sub-BVHs based on specific *change categories*. This allows us to optimally determine when to use reuse, refit, or rebuild operations for maximum update efficiency. Our approach supports high-performance, interactive GPU ray tracing of complex production scenes, maintaining fluid updates even during real-time animation playback or active geometry editing by an artist.



SIGGRAPH Talks '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2541-8/2026/07
https://doi.org/10.1145/3799818.3812105

CCS Concepts

• **Computing methodologies** → **Ray tracing**.

ACM Reference Format:
Andrew Bauer and Yining Karl Li. 2026. A Hybrid BVH Structure for Interactive GPU Ray Tracing. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH Talks '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3799818.3812105

1 BVH Structure and Performance

For any given 3D scene, there is an unbounded number of ways to construct a BVH over all objects in the scene for the purpose of accelerating ray intersection tests. The specific structure and topology of a BVH significantly impact the performance characteristics of the BVH. Often, a key tradeoff to consider is optimizing for BVH construction/update speed versus optimizing for BVH traversal speed; these two goals can often be in opposition to each other. Therefore, choosing a particular BVH structure when building a high-performance interactive renderer requires carefully considering the needs of the target application and artist workflows.

1.1 Two-Level BVH

One common strategy is to build a bottom-level acceleration structure (BLAS) for each scene object, then join each BLAS into a single scene-level instance acceleration structure (IAS) [Parker et al. 2010;

Wald et al. 2014]. This two-level BVH structure [Wald et al. 2003] is useful because it is simple, efficient to traverse, and uses relatively little memory when nested instancing is minimal. This BVH structure is also fairly efficient to update in response to scene changes; when a scene object's geometry changes it is only necessary to rebuild or refit that BLAS, then refit the scene-level IAS.

However, this two-level structure does not consider the complexity of individual scene objects; if a scene is segmented into a large number of very low-complexity scene objects, then the two-level BVH structure may suffer from performance issues due to per-BLAS overhead. Additionally, while a two-level BVH is typically very efficient to traverse, a significant amount of overlap between BLASes will result in reduced traversal performance. While this problem can be ameliorated using techniques such as rebraiding [Benthin et al. 2017], as we do in our offline production renderer Hyperion [Burley et al. 2018], rebraiding adds additional complexity to the BVH build and update process, posing further challenges for interactive applications.

1.2 Mega-BLAS

To address the issues with the two-level BVH structure, one might consider a drastic change in strategy. We can minimize both the per-BLAS overhead and the overlapping BLAS ray traversal overhead simply by merging all scene object geometry into a single scene-level BLAS. This extreme BVH structure, which for convenience we call a *Mega-BLAS*, can be surprisingly efficient to build on the GPU (such as with the HLBVH family of BVH builders [Garanzha et al. 2011; Meister et al. 2021; Pantaleoni and Luebke 2010]), and when a large portion of the scene geometry deforms all at once it can be very efficient to refit an existing Mega-BLAS. Additionally, because it does not impose any external structure or overlapping bounds on the scene geometry, the Mega-BLAS has exceptional ray traversal performance.

However, there are significant performance tradeoffs when comparing Mega-BLAS to the two-level approach. If a single scene object changes, the two-level approach only requires rebuilding that object's BLAS and the scene-level IAS. Under Mega-BLAS, however, a single change requires rebuilding the entire structure from scratch [Wald and Havran 2006]. A similar performance disparity exists for operations like adding, removing, transforming, and deforming scene objects.

Memory footprint is also a significant concern with a Mega-BLAS. While the two-level BVH structure supports the creation of many instances of a single scene object with very little memory, the Mega-BLAS requires the duplication of a scene object's geometry for each instance of the scene object. This memory usage quickly adds up when hundreds, thousands, or more instances are present, particularly on a memory-constrained GPU. Additionally, a large amount of temporary memory is required when building a Mega-BLAS, whereas a much smaller amount can be reused to build many smaller scene object BLASes. Although this temporary memory may be freed after the BVH is built, it can significantly increase the memory high-water mark, which reduces the maximum supported scene complexity.

2 Our Hybrid Approach

Disney Animation utilizes an in-house interactive GPU-based ray tracer for preview rendering in DCC viewports to accelerate various artist workflows. Our interactive renderer produces a higher quality result than traditional rasterized viewports, but trades off matching final frame fidelity in exchange for interactive speed. In particular, the ability to respond to live scene edits and animation playback in real-time can dramatically improve artist productivity.

We previously used a three-level BVH, similar to the two-level BVH structure but with an added intermediate level of IASes to spatially segment the scene object BLASes. This spatial segmentation removed the need to refit or rebuild a very large scene-level IAS on small scene changes. Although this structure had very good scalability to a huge number of instances, it otherwise maintained the same update and traversal performance as a two-level BVH.

To improve traversal performance and handle large-scale changes, we turned to the Mega-BLAS, but we first needed to address its significant weaknesses in order to make it a practical solution. As a result, we have designed a hybrid approach combining the strengths of the two-level and Mega-BLAS approaches while minimizing the weaknesses of each.

2.1 Memory-Efficient Instancing

The two-level BVH structure is very efficient at representing many instances with very little memory while maintaining good ray traversal performance. For this reason, our interactive ray tracer uses a two-level BVH structure for all scene objects with two or more instances. All the remaining non-instanced scene objects are placed into one of a fixed number of Mega-BLASes, and a single instance of each of these Mega-BLASes is added to the scene-level IAS alongside the individual scene object instances. This hybrid BVH structure retains low memory utilization for scenes with significant instancing, but is still able to take advantage of the performance benefits of Mega-BLASes.

2.2 Change Category Binning

To address the performance issue where a change in a single scene object triggers a rebuild of a large Mega-BLAS, we partition the non-instanced scene objects based on their change category. An object's change category is typically a strong indicator of which future changes are likely for the object, and thus what operations may be necessary to update any BLAS which includes it. In particular, a BLAS refit is necessary if a scene object's vertices (and/or similar non-topological data, such as radii for sphere geometry) or topology changes. Scene description formats often contain some information that can be used to infer a change category, but in practice this information may not be applicable to every workflow, and is often inaccessible by the renderer. Instead of relying on authored hints, we partition scene objects into three change categories based on their change history in the current session:

(1) **Fully-Static**: The scene object's geometry has never changed since creation. This category is the default for new objects.
(2) **Dynamic-Vertex**: The scene object's vertices (and/or similar non-topological data) have changed at some point since creation. The containing BLAS must be either refit or rebuilt.

(3) **Dynamic-Topology**: The scene object's topology has changed. The containing BLAS must be rebuilt.

After scene object partitioning, we build one Mega-BLAS per change category. While a change category shift necessitates a slow rebuild of both involved Mega-BLASes, this occurs rarely since change categories typically reliably predict future change patterns. This grouping limits refits to the Dynamic-Vertex Mega-BLAS and rebuilds to the Dynamic-Topology Mega-BLAS. We also use change categories to inform build flags in OptiX [Parker et al. 2010]. The Fully-Static category enables *PREFER_FAST_TRACE* and *ALLOW_COMPACTION* while disabling *ALLOW_UPDATE*. Dynamic-Vertex enables all three, whereas Dynamic-Topology disables all three and instead prioritizes *PREFER_FAST_BUILD*. These flags fine-tune the balance between update speed, traversal performance, and memory consumption.

Note that the resulting Mega-BLASes are often each approximately the size of the full scene. As previously discussed, overlapping BLASes is known to degrade BVH traversal performance. Additionally, techniques such as rebraiding [Benthin et al. 2017] are not applicable to a set of Mega-BLASes. In practice, the improvements to traversal performance from using Mega-BLASes more than makes up for the penalty for significant overlap between the fixed set of Mega-BLASes.

3 Results

Office Scene

	Two-level	Hybrid (Ours)
frame integration time (ms)	5.73	5.35
frame update time (ms)	46.16	15.45
memory usage (MiB)	99.87	93.31
memory high-water mark (MiB)	99.87	196.51

Can Throw Scene

	Two-level	Hybrid (Ours)
frame integration time (ms)	7.80	6.75
frame update time (ms)	33.31	22.36
memory usage (MiB)	124.84	95.62
memory high-water mark (MiB)	152.65	231.97

Marsh Market Scene

	Two-level	Hybrid (Ours)
frame integration time (ms)	3.29	3.25
frame update time (ms)	39.17	11.68
memory usage (MiB)	3282.59	1938.81
memory high-water mark (MiB)	3285.61	7131.22

We compared the two-level BVH structure to our hybrid BVH structure across three scenes from "Zootopia 2", using several metrics. These scenes are representative of the scene as visible to artists in a DCC viewport, but do not have the complete scene data that is available during final frame rendering. Each scene is rendered at a resolution of 2283×1215. "Frame integration time" represents the path tracing portion of a single frame, excluding scene processing (such as BVH updates) and post-processing (such as denoising). "Frame update time" represents only the scene processing after the DCC moves to a new frame. Memory usage and memory high-water mark refer to the GPU memory used to store BVH-related data, including temporary memory required for a BVH build/refit, but excluding independent copies of vertex and topology data.

4 Conclusions and Future Work

Our hybrid BVH maintains the scalability of a two-level BVH while capturing the performance advantages of the Mega-BLAS. Our artists report smoother workflows with fewer interruptions during scene editing. While our approach increases GPU memory usage and could eventually approach OptiX primitive limits for massive geometry, these factors have not yet impeded us in practice.

Notably, our hybrid structure may incur a brief 1-frame delay when an edit triggers a change-category shift. This delay is acceptable for animation workflows, but may be less suitable for strict real-time applications such as games or virtual production.

For future work, we are interested in further tempering the disadvantages inherited from the Mega-BLAS. One interesting idea is spatial segmentation; creating smaller, localized Mega-BLASes could reduce memory overhead, minimize the 1-frame delay, and help avoid primitive count limits; we have already explored this idea with promising results in a predecessor to our hybrid system.

Currently, Mega-BLASes are restricted to meshes. All curves (typically hair, curve-based garments [Lipson and Velasquez 2023; Velasquez et al. 2022], or animated vegetation) default to a standard two-level BVH. This avoids curve-merging challenges and provides an effective performance fallback for non-static geometry spanning large regions. While this approach works, improving curve handling remains a high-priority research goal for our future development.

References

Carsten Benthin, Sven Woop, Ingo Wald, and Attila A. Áfra. 2017. Improved Two-Level BVHs using Partial Re-Braiding. In *Proc. of HPG*. Article 8, 7:1–7:8 pages. https://doi.org/10.1145/3105762.3105776

Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Dan Teece. 2018. The Design and Evolution of Disney's Hyperion Renderer. *ACM Transactions on Graphics* 37, 3, Article 33 (Aug. 2018). https://doi.org/10.1145/3182159

Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. 2011. Simpler and Faster HLBVH with Work Queues. In *Proc. of HPG*. 59–64. https://doi.org/10.1145/2018323.2018333

Dan Lipson and Jose Velasquez. 2023. Creating Curve-based Garments With Custom Weave Patterns. In *ACM SIGGRAPH 2023 Talks (SIGGRAPH '23)*. Article 18, 2 pages. https://doi.org/0.1145/3587421.3595443

Daniel Meister, Shinji Ogaki, Carsten Benthin, Michael J. Doyle, Michael Guthe, and Jiri Bittner. 2021. A Survey on Bounding Volume Hierarchies for Ray Tracing. *Computer Graphics Forum (Proc. of Eurographics)* 40, 2 (May 2021), 683–712. https://doi.org/10.1111/cgf.142662

Jacopo Pantaleoni and David Luebke. 2010. HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. In *Proc. of HPG*. 87–95. https://doi.org/10.2312/EGGH/HPG10/087-095

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics (Proc. of SIGGRAPH)* 29, 4, Article 66 (July 2010). https://doi.org/10.1145/1778765.1778803

Jose Velasquez, Alexander Alvarado, Ying Liu, and Maryann Simmons. 2022. Embroidery and Cloth Fiber Workflows on Disney's "Encanto". In *ACM SIGGRAPH 2022 Talks (SIGGRAPH '22)*. Article 22, 2 pages. https://doi.org/10.1145/3532836.3536250

Ingo Wald, Carsten Benthin, and Philipp Slusallek. 2003. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proc. of IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. 77–86. https://doi.org/10.1109/PVGS.2003.1249045

Ingo Wald and Vlastimil Havran. 2006. On Building Fast kd-Trees for Ray Tracing, and On Doing That in O(N log N). In *Proc. of IEEE Symposium on Interactive Ray Tracing*. 61–69. https://doi.org/10.1109/RT.2006.280216

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (Proc. of SIGGRAPH)* 33, 4, Article 143 (Aug. 2014), 143:1–143:8 pages. https://doi.org/10.1145/2601097.2601199